
Let $t(x)$ be the number of steps taken by algorithm $A$ on input $x$. Let $T(n)$ be the worst-case time complexity of algorithm $A$:

$$T(n) = \max_{x \text{ is an input of size } n} t(x) = \max \{t(x) : x \text{ is an input of size } n\}$$

1. To prove that $T(n)$ is $O(g(n))$, one must show that there is a constant $c > 0$, and an input size $n_0 > 0$, such that for all $n \geq n_0$:

   $$T(n) \leq c \cdot g(n)$$

   $\Leftrightarrow$ max \{t(x) : x is an input of size n\} $\leq c \cdot g(n)$

   $\Leftrightarrow$ For every input $x$ of size $n$, $t(x) \leq c \cdot g(n)$

   $\Leftrightarrow$ For every input of size $n$, $A$ takes at most $c \cdot g(n)$ steps

2. To prove that $T(n)$ is $\Omega(g(n))$, one must show that there is a constant $c > 0$, and an input size $n_0 > 0$, such that for all $n \geq n_0$:

   $$T(n) \geq c \cdot g(n)$$

   $\Leftrightarrow$ max \{t(x) : x is an input of size n\} $\geq c \cdot g(n)$

   $\Leftrightarrow$ For some input $x$ of size $n$, $t(x) \geq c \cdot g(n)$

   $\Leftrightarrow$ For some input of size $n$, $A$ takes at least $c \cdot g(n)$ steps

In summary:

Let $T(n)$ be the worst-case time complexity of algorithm $A$.

1. $T(n)$ is $\tilde{O}(g(n))$ iff $\exists c > 0, \exists n_0 > 0$, such that $\forall n \geq n_0$:

   for every input of size $n$, $A$ takes at most $c \cdot g(n)$ steps.

2. $T(n)$ is $\Omega(g(n))$ iff $\exists c > 0, \exists n_0 > 0$, such that $\forall n \geq n_0$:

   for some input of size $n$, $A$ takes at least $c \cdot g(n)$ steps.

3. $T(n)$ is $O(g(n))$ iff $T(n)$ is $O(g(n))$ and $T(n)$ is $\Omega(g(n))$. 

...
**DELETE_MIN(T)**

**Example:**

\[ T = \]

```
    11
   / \  
  15  22
  /   /  
 18  13  23
   
 21
```

**Delete_MIN(T):**

1. Scan roots to find smallest element.
2. Delete element.
3. Merge resulting BQ's.

**Carry:**

```
    23
   / 
  22 23
   /   
 27  13
   
 14
```

\[ T_1 = T - S_3 \]

\[ T_2 = S_3 - \{x\} \]

\[ T = \text{UNION}(T_1, T_2) \]
Key comparisons
8-5 = 3
Merge heap
8 edges
Total of
5 edges
Example of Binary Tree Merge

Example of Binary Tree Merge
<table>
<thead>
<tr>
<th>ADT</th>
<th>Description</th>
</tr>
</thead>
</table>
| Priority Queue | Set S with keys/kys  
|             | insert (S, k)  
|             | max (S)  
|             | extract max (S)  |
| Mergeable Queue | union (S, T)  |
| Dictionary  | Set  |

<table>
<thead>
<tr>
<th>DT</th>
<th></th>
</tr>
</thead>
</table>
| Unordered LL | ins \( \Theta(1) \)  
|             | ex. max \( \Theta(-) \)  |
| Ordered LL  | ins \( \Theta(-) \)  
|             | ex. max \( \Theta(1) \)  |
| Heap | ins, ex. max \( \Theta(1/n) \)  |
| Binomial Heap | ins \( \Theta(\log n) \)  
|             | ex. min \( \Theta(\log n) \)  
|             | union \( \Theta(\log n) \)  |
| BST         | \( \Theta(n) \)  |
|           | balanced BST  
|           | to AVL-tree  |
\( BH \in H \) \( BH = \min BH \) \( \max BH = H = \min H \) \( \Rightarrow H = H \)
**Data structures**

- **ADT**: description of object and ops
- **DS**: specific implementation of ADT

  - e.g. Priority Queues
    - obj: set S of donor w/ keys
    - ops:
      - `insert((s, x))`: \( S \cup \{x\} \)
      - `max(S)`: \( \rightarrow \max(S) \)
      - `extract-max(S)`: \( S \leftarrow S - \{\max(S)\} \)

  - e.g. unsorted linked list
    - ordered \( \nRightarrow \)
    - heap

**CBT**:

- **Max-Heap**: CBT, s.t. \( \forall m \in \text{CBT}, \text{value}(m) \geq \text{value}(\text{child}(m)) \)
- **DT**:
  - height of tree is length of longest path from root to leaf
  - heap ops: 1. Get CBT shape right
    - constraints 2. Get near heap right
      - insert: put at: lost pos. then swap \( \text{max} : \text{height} \)
      - `extract-max`: swap last pos with root
      - decrease heap size
      - heapify (swap, compare w/ heaupsize)
### ADT

**Priority Q**
- priority Q
- mergeable priority Q

### Ops
- `insert`
- `max`
- `extract-max`
- All above + Union (S, T)

### DS
- **Heap (CERS6)**
- **Binomial Heaps**
  - based on bin-forest

### CBT

### Sk-Trees

### Sk-Tree

A Sk-Tree has $2^k$ nodes and $\binom{k}{d}$ nodes at depth $d$.

### Bimomial Forest

A bimomial forest of size $m$, $(BF_m)$ is a sequence of $BF_k$ with $m = <b_0, b_1, b_2, ..., b_k>$ nodes

- $BF_{<0,0>} = <$ all Sk-trees s.t. $b_0 = 1>$
- $BF_{<2,1,1,1>} = <$ all Sk-trees s.t. $b_0 = 1, b_1 = 2, b_2 = 3, b_3 = 4, b_4 = 1>$

- A BFM with $m = <b_0, b_1, ..., b_k>$ nodes $BF_m = <$ all Sk-trees s.t. $b_k = 1>$
  - Largest tree in BFM is $S_k$ with $k = \lfloor \log m \rfloor$
  - Let $\alpha(m) = \#$ of 1's in the binary rep of $m$
    - BFM has $\alpha(m)$ trees
    - BFM has $m - \alpha(m)$ edges

A (min) binomial heap of $m$ elements with keys is a BFM s.t.

- Each node of BFM stores 1 element
- Each Sk-tree of BFM is (min) heap-ordered

### Example

- $S_1 → 0$
- $S_2 → 1$
- $S_3 →$
  - $S_4 →$
1. \( S \leftarrow \text{UNION} (S, T) \rightarrow \text{binary add}. \)

2. \( \text{insert} \rightarrow \text{before} \{ x \} \rightarrow S_0; \emptyset \rightarrow \text{UNION} \)

3. \( \text{MIN}(T) \rightarrow \text{look at roots} \quad \Theta (\log n) \)

4. \( \text{Extract-min} / \text{Delete-min} \)

**Example:** \( |T| = 27 \rightarrow \{ s_4, s_3, s_1, s_0 \} \quad (11011) \)

- 27 - 4 = 23 edges
- \( \text{insert} \{ x \} : \{ 11011 \} \rightarrow 2 \text{ key comp} \rightarrow 2 \text{ new edges} \)
- \( \rightarrow 25 \text{ edges} \) (check: 28 - 3 = 25)

- Bin heap: since based on bin add.
- When \( \text{insertions} > \log n \rightarrow \text{avg. insert is } O(1) \text{ less.} \)

**Priority Queues** implemented with heaps (and used almost always for that purpose).
ADT | OPS
---|---
Dictionary | Insert, Delete, Search

Data Structures
- Balanced BST
  - 2-3 trees
  - Red-black trees
  - AVL trees

\[ BF(v) = h_R - h_L \]

AVL tree: BST s.t. for every AVL, \( BF(v) = -1, 0, 1 \)

height of AVL: with \( n \) nodes, have height \( \Theta(\log n) \)

- Can do inserts and deletes while maintaining the balance

\[ \text{insert}(T, x) \]
  - inserting as in any BST, \( x = \text{leaf} \)
  - if encountering \( O \), need to go up any more
  - rotate if needed

After Rotation:
1) Relabeled
2) Balance preserved
3) Height same as before (towers)
Rebalancing

Let $A$ be the first node on path to root($t$) that becomes unbalanced.

W.l.o.g. $A + 1 \leq 2$ (spn: $A - 1 \geq 2$)

- case (1).1

- case (1).2

Deletion

No bonus

Vehicle on $\,$ Sortling + MS

$236 \, \text{SLOG} \rightarrow 263 \, \text{SLOG} \,$

TCP
Augmenting Data Structures

E.g. Dynamic Order Statistics

- set S of m elements with keys
- ops: insert, delete, search
  - select (k): return the element of rank k
  - rank (v): returns the rank of v

\[ S = \{5, 15, 27, 30, 563\} \]
\[ \text{select (4)} = 30 \]
\[ \text{rank} (15) = 2 \]

\[ \text{size (v)} = \text{size (left (s))} + \text{size (right (s))} + 1 \]

Relative Ranking

\[ \text{RR (x)} = \text{size (left (x))} + 1 \]

select (k) \(\Rightarrow\) to find kth rank,
if \(k < \text{RR (x)}\), find kth rank of left (x), ok.

\(\Rightarrow\) (since AVL tree), select (k) takes \(O(\log n)\) in WTC.

rank (k) \(\Rightarrow\) determine RR (x) in k-subtree.
in every mode from path x \(\rightarrow\) root, determine RR (x) in y-subtree:

- RR (y') = RR (x) if x \(\rightarrow\) y' (same)
- RR (x) + RR (y) if x \(\rightarrow\) y

\(\Rightarrow\) (\(\Rightarrow\)), rank (k) takes \(O(\log n)\) in WTC.
In order to maintain size info \( \frac{k}{n} \):

1. \( k \) ends up a leaf \( \Rightarrow \) size\((x) = 1 \)
2. add 1 to size\((y)\) for all \( y \) on path \( x \rightarrow \text{root} \)
3. proceed (as usual in AVL) from \( x \rightarrow \text{root} \)
4. updating balance factors and rebalancing

\[ \Rightarrow \text{ takes } O(\log n) \]

### Average / Expected Complexity

**Example:** implementing a dictionary with keys \( K \subseteq U \), \( U = \{a, b, \ldots , u-1\} \)

1. \( u \) is small, use a direct access table (e.g., array)
2. \( u \) is very large, use a hash table

   - let \( k \in U \) and \( T \) ("hash table") \( |T| = m \) (i.e., \([0, m-1]\)
   - hash function \( h : U \rightarrow T \)
     \[ e \rightarrow h(e) = i, \ i \in [0, m-1] \]

**Hashing w/ Chaining:**

\[ H \rightarrow T \rightarrow \text{LIFO} \]

**UC** is \( O(n) \) (if \( h() \) leads to many collisions)

### Simple Uniform Hashing Assumption (SUHA)

Any key \( k \) is equally likely to hash into any of \( m \) slots \( U \).

**Assumption relies on** \( h() \) and dist. of \( K \subseteq U \).

\[ P(h(k) = i) = \frac{1}{m} \quad (i \in [0, m-1]) \]

**Proof:**

\[ E(m_0 + \ldots + m_{m-1}) = \frac{m^{m-1}}{m} (E(m_i)) \]

By SUHA, \( \forall i, j \), \( E(m_i) = E(m_j) \)

**Proof:**

\[ m \cdot E(m_i) = m \cdot E(m_{i-1}) \]

\( \Rightarrow \) \( m \cdot E(m_i) \leq \alpha \cdot m \) \( \Rightarrow \) \( \alpha \leq 1 \)

Load factor
Worst case Cost of RQS(S) is $\Theta(n^2)$

**Analysis of RQS**
- 2 keys compared if $i$ is a pivot
- 2 keys can compare at most once
- 2 keys split apart by pivot are never compared

Fix $|S| = n$. Cost of RQS = $n$ key comps.

WC: $C = \Theta(n^2)$
AC: $E(c) =$ ?

$S$ has $n$ keys: $z_1, z_2, \ldots, z_i, \ldots, z_k, \ldots, z_n$

$e_{ij} = \begin{cases} 1 & \text{if } z_i \text{ comp to } z_j \\ 0 & \text{otherwise} \end{cases}$

$$\Rightarrow E(c) = E\left[ \sum_{i,j} e_{ij} \right] = \sum_{i,j} E(e_{ij})$$

$E(e_{ij}) = 1 \times p(e_{ij}) + 0 \times p(e_{ij}^c) = p(e_{ij})$

$$E(c) = \sum_{i,j} p(e_{ij}) = \sum_{i=1}^{n} \sum_{j=1}^{n} p(e_{ij})$$

Consider $Z_{ij} \subseteq Z$ (sorted $S$), $|Z_{ij}| = j-i+1$

- Initially $Z_{ij} \subseteq Z = S$
- RQS keeps selecting pivots
  - as long as new aren't in $Z_{ij}$, $Z_{ij}$ remains
  - at some pt, $RQS(S)$ must select a pivot within $Z_{ij}$
- Case 1: $z_i < p(z_j) \Rightarrow z_i$ is never comp to $z_j$
- Case 2: $p \in (Z_i \cup Z_j) \Rightarrow \text{pick f that is} \frac{2}{j-i+1}$

$$\Rightarrow E(c) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n} \sum_{j=i}^{n} \frac{2}{j+1}$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{n-i} \frac{2}{k+1} \leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k}$$

$$= 2 \times \sum_{i=1}^{n} \frac{1}{i} \times \sum_{k=1}^{n} \frac{1}{k}$$

$$\Rightarrow E(c) \leq 2mH_m \land H_m \in O(\log n)$$

$E(c) \in O(n \log n)$
Hashing ch4d

SUHA: \( V \subseteq j \in \{0, 1, \ldots, m - 1\}, Pr[h(k) = j] = \frac{1}{m} \)

1. \( k \notin T \): \( E = \alpha \)
2. \( k \in T \): \( E \propto \frac{1}{2} \)

D Assumed that \( Pr[k = k_i] = \frac{1}{m} \)

"It is equally likely to be any of the \( m \) keys inserted in \( T \)."

\( \Rightarrow \) Expected cost: \( E = \frac{1}{m} \sum \frac{m-1}{m}^i \frac{1}{m^{i+1}} \), \( \Rightarrow \) if \( k \) was \( i \)th key to be inserted, expected # comp: \( \frac{1}{m} \sum \frac{1}{m^{i+1}} \) \( \Rightarrow \) expected # comp of search(k) = \( \frac{1}{m} \sum \frac{1}{m^{i+1}} \) \( = N \) - connecting factor from \( \binom{i}{2} \)

E.g. \( m = 2000 \)
- \( p = 0.1 \) is prime
- \( \alpha' \leq 3 \)
- \( m \geq \frac{2000}{3} \)
- \( m > 666 \Rightarrow m = 701 \)

D Probabilistic algorithm
- Assumes inputs is random - no input follows some 'nice' dist., e.g., hashing - SUHA

D Randomized algorithm
- Uses non-determinism

\( \text{Randomized Quick sort (recursive guy)} \)

Input: set of \( n \) distinct keys
Output: sorted keys
\( RQS(S) \). If \( S = \emptyset \), then return.
- If \( |S| = 1 \), then output \( k \) and return.
- If \( |S| \geq 2 \), then select pivot \( p \) at random.
- By comparing \( p \) with every other key in \( S \), split \( S \) into \( S_L = \{ s \in S \mid s < p \} \), \( S_R = \{ s \in S \mid s \geq p \} \).
- \( RQS(S_L) \), output \( p \), \( RQS(S_R) \)
Disjoint Sets

- In disjoint elements, initially each is in its own set: \( S_1 = \{ 1 \}, S_2 = \{ 2 \}, \ldots, S_m = \{ m \} \)
- Each set has a representative
  \( S_x = \{ x \} \) set rep. by \( x \)
- Operations:
  + \text{UNION}(S_x, S_y) \): Replaces sets \( S_x \) and \( S_y \) by \( S = S_x \cup S_y \)
  + \text{FIND}(x) \): Finds \( S \) s.t. \( x \in S \)
    \[ \text{replaces } \text{rep} \text{ to } \text{set rep} \text{ of } S \]

- At most \( m-1 \) \text{UNIONS} possible
- \( \sigma = \text{seq of } m-1 \text{ UNIONS} \)
  Mixed w/ \( m \geq m \) finds
  - How to minimize the cost of executing \( \sigma \) seqs?
    - Implement: linked list where head is the rep of the set
      - Cost of \text{UNION} \( : O(1) \) \( \times m \)
      - Cost of \text{FIND} \( : O(1) \) \( \times m \)

- Augmented linked list
  - Cost: \( O(m) \) \( \times m \)
    - \( O(m^2) \)
  - Weighted \text{UNION} rule
    - Analysis in Tutorial

- Forest DT
  - Cost: \( O(1) \)
  - Cost: \( O(1) \)
  - How to reduce?

- Weighted \text{UNION} [by size]

- Lemma: With \text{UNION} rule (by size)
  - Any tree of height \( h \)
  - \text{consolidated during execution of seq} \( \sigma \)
  - Has at least \( 2h \) nodes, i.e. \( m \geq 2h \)
Thus, by Lemma 2k, \( ||f||_2 \leq 2^{k+1} \), and hence

\[
\|\mathbf{W}\|_2 \leq 2^{k+1} \|\mathbf{W}\|_F^n = 2^{(k+1)n} \|\mathbf{W}\|_F
\]

will be small.

Proof

Consider the following note: the norm of a matrix is the largest sum of absolute values of its entries.

By the triangle inequality, we have

\[
\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F^n = \sum_{i,j} |a_{ij}|^n = \sum_{i,j} |a_{ij}| = \|\mathbf{A}\|_2
\]

for any matrix \( \mathbf{A} \).
Dynamic Table

(2) Expansion: if $\alpha(T) = 1$ and insert occurs, size(new $T$) = $2\times$ size($T$)

(b) Contraction: if $\alpha(T) = \frac{1}{2}$ and delete

\[ \alpha(T) = \frac{1}{2} \]

\[ T \quad \rightarrow \quad \text{new } T \]

\[ T \quad \rightarrow \quad \text{new } T \]

**Example**

- Each insert is charged $3$
  - $1$ $\$$ actual cost
  - $2$ $\$$ credit

- Each delete $2$ $\$$

- If $\alpha$ occurs, then at least $\frac{m}{2}$ inserts occurred.
  - These generated a total of $\left(\frac{m}{2}\right) \times 2$ $\$$ = $m$ $\$$ of credit.

- $\frac{m}{2}$ deletes occurred.
  - $\left(\frac{m}{2}\right) \times 1$ $\$$ = $\frac{m}{2}$ $\$$

- Cost: $\text{const}$
Graphs \( G = (V, E) \), \( |V| = n \), \( |E| = m \)

Adjacency list

\[
\begin{array}{c|c|c|c}
\text{Vertex} & \text{Edges} & \text{Degree} \\
\hline
0 & 2 & 1 \\
1 & 3 & 2 \\
2 & 3 & 2 \\
3 & 0 & 0 \\
4 & 0 & 0 \\
5 & 0 & 0 \\
6 & 0 & 0 \\
\hline
\end{array}
\]

- Adjacency list

\( O(m + n) \)

\( O(m) \)

**DFS**

1. Start \( s \in V \), explore all neighbours
2. Colour \( (v) \) = white if \( v \) is undiscovered
3. Grey if \( v \) was disc. but not explored
4. Black if \( v \) was disc and expl.
5. Length of disc. path \( : d[v] \)
6. \( P[v] = u \) if \( u \) disc. \( v \)
7. \( d[v] = d[u] + 1 \)

- DFS tree is graph induced by all edges \((u,v)\) s.t. \( P[v] = u \)

**BFS**

\( d[v] \) = disc. \( v \)

**BFS alg**

- Initialization
  
  - colour [s] = grey, \( d[s] = 0 \), \( P[s] = Nil \)
  
  - \( V \setminus \{s\} \), colour \( [v] \) = white, \( d[v] = \infty \), \( P[v] = Nil \)
  
  - \( Q \) = empty, \( ENQ(Q, s) \)

- Code
  
  - While \( Q \neq \emptyset \)
    
    - \( u \leftarrow DEQ(Q) \)
      
      - For each \((u,v) \in E\)
        
        - If colour \( (v) = \) white then
          
          - Colour \( (v) = \) grey
          
          - \( P[v] = u \)
          
          - \( d[v] = d[u] + 1 \)
          
          - \( ENQ(Q, v) \)

- \( \text{LCV complexity: } O(m+n) \)

- Let \( d(s,v) = \) length of shortest path starting from \( s \) to \( v \),

\( \Rightarrow d(s,v) \leq d[v] \)
The complexity of BFS (graph - dic.) is \( O(\text{m+n}) \)

Let \( \delta(s, u) \leq d[u] \)

If \( u \) enters \( Q \) before \( v \) during BFS \( (s) \), then \( d[u] \leq d[v] \)

Proof: Suppose \( \neg \delta \) for contradiction that \( \neg \delta \).

Let \( v \) be the 1st node that enters \( Q \) s.t. \( d[u] > d[v] \) for some \( u \).

\( u \) and \( v \) entered \( Q \) during exploration of some nodes \( u' \) and \( v' \) resp.

\( d[u] = d[u'] + 1 \)
\( d[v] = d[v'] + 1 \)
\( \neg \delta \)

Since \( u \) entered \( Q \) before \( v \), \( u' \) was exp. before \( v' \), \( u' \) entered \( Q \) before \( v' \) entered \( Q \).

By def., \( d[u'] \leq d[u'] + 1 \)
\( d[u'] \leq d[u'] + 1 \)

Theorem: After BFS \( (s) \), \( \forall u \in V \), \( d[u] \leq \delta(s, u) \)

Proof: Suppose for contradiction, \( \exists x \in V \), \( d[x] > \delta(s, x) \)

Let \( v \) be closest node from \( s \) s.t. \( d[v] > \delta(s, u) \).

Consider a shortest path \( s \) to \( v \)

\( \delta(s, v) = \delta(s, u) + 1 \)

Since \( u \) is closer than \( v \) to \( s \), it must be that \( d[u] = d[u] + 1 \)

- Consider the colour of \( u \); just before \( u \) is expl.
  - (a) \( u \) is white: \( u \) expl before \( v \)
  - (b) \( u \) is black: \( u \) was expl. before \( v, v \) expl. \( Q \) before \( u \)

By \( \Delta \) \( \Rightarrow d[u] \leq d[u] \)
(c) \( V \) is grey/gray, some \( w \) disc. \( v \) bef. \( u \) is expl.

\[ \text{w expl. bef u expl.} \]
\[ \Rightarrow \text{w enters } Q \text{ bef } u \text{ enters } Q \]
\[ \Rightarrow d[w] \leq d[w] \]
\[ d[u] \leq d[u] + 1 \]
\[ \Rightarrow d[v] = d[u] + 1 \]

---

**Depth-first Search**

Same as in BFS: colours, \( P[v] = u \) iff \( u \) disc. \( v \)

**DFS**

Different from BFS: global var \( f \) for time (counter)

\[ d[u] \text{: time when } u \text{ was disc} \]
\[ f[u] \text{: } u \text{'s expl. was compl.} \]

**alg. DFS**

\[ \text{DFS}(G) \]

\[ \text{[For each } u \in V \text{]} \]
\[ \text{colour}[u] \leftarrow w; \quad d[u] \leftarrow \infty; \quad f[u] \leftarrow \infty; \quad P[u] \leftarrow \text{NIL} \]
\[ \text{time} \leftarrow 0 \]
\[ \text{[For each } v \in V \text{]} \]
\[ \text{if colour}[v] = w, \text{then DFS-explore}(G, v) \]

**DFS-explore**

\[ \text{colour}[u] \leftarrow \text{grey} \]
\[ \text{time} \leftarrow \text{time} + 1; \quad d[u] \leftarrow \text{time} \]
\[ \text{[For each edge } (u, v) \in E \text{]} \]
\[ \text{if colour}[v] = \text{white then} \]
\[ P[u] \leftarrow u \]
\[ \text{DFS-explore}(G, v) \]
\[ \text{colour}[u] \leftarrow \text{black} \]
\[ \text{time} \leftarrow \text{time} + 1; \quad f[u] \leftarrow \text{time} \]

\[ \Rightarrow \text{DFS forest} \]
Claim 1: \( r \) is a desc. of \( u \) in BFS forest iff
\[ d[u] < d[r] < f[r] < f[u] \]

Claim 2: For any 2 nodes, \( u \) and \( r \), we cannot have
\[ d[u] < d[r] < f[r] < f[u] \]
overlapping dist. intervals are imp.

Claim 3: If \((u, v) \in E\), then \( d[v] < f[v] \)

Application of BFS: use to find # comp. (and answer)
- No cycles \( \Rightarrow |E| = |V| - #\text{ comp.} \)

GBM - o

SD Hypercube

\[ \sum_{i=1}^{n} s_i = n \]
\[ s_i \geq 0 \]
\[ \sum_{i=1}^{n} s_i = 2^n - 1 \]
A DFS of a directed graph $G$ classifies its edges as follows:

1. True edge: iff $u$ is parent of $v$ in DFS of $G$
2. Back edge: iff $u$ is an ancestor of $v$ otherwise
3. Cross edge: iff $u$ is a descendant of $v$

Claim: $(u,v) \in E$ is of type

- 0 or 1 iff $d[u] < d[v] < f[v] < f[u]$
- 3 iff $d[v] < f[v] < d[u] < f[u]$

Application of DFS

- White Path Theorem (CLRS 27.3)
  
  $\forall \in A $ DFS of $G$, $v$ becomes a descendant of $u$ iff
  
  at the same time $d[v] > f[u]$ (the DFS disc. $u$). There is a path from $u$ to $v$ in $G$ that consists entirely of white nodes.
**Proof**

1. Suppose $v$ is a descendant of $u$ in the DFS at $d[v]$: not yet discovered so they are all white.

2. Suppose at the line $d[w]$, suppose for contra, that $v$ does not have a descendant of $u$. Let $w$ be the closest such node to $w$ in that path. Then $w = u$ or $w$ is descendant of $u$.

   - $d[w] < d[z] \Rightarrow z$ is white at line $d[w]$.
   - $d[z] < f[u] \Rightarrow y \in S$.
   - $f[u] \leq f[w] \Rightarrow$ because $w = u$ or $w$ desc. of $u$.

   \[ \Rightarrow d[w] < d[z] < f[z] < f[u] \]

   \[ \Rightarrow z \text{ is disc during exp. interval (d[w], f[u])} \]

   \[ \Rightarrow z \text{ is a descendant of } u \]

   \[ \Rightarrow \text{ contra!} \]

Thus, $v$ is a descendant of $u$. 

**Theorem (CLRS 22.11)**

For any directed graph $G$, the DFS of $G$ has a back edge:

- A directed $G$, A DFS of $G$:
  - $G$ has a cycle $\iff$ DFS of $G$ has a back edge.

**Proof**

1. Suppose DFS of $G$ finds a back edge $(v, u)$ then $\Rightarrow$ $G$ has a cycle.

2. Suppose $G$ has a cycle $C$.

   - Let $u$ be the first node in $C$ DFS discovers.
   - Let $v$ be the guy before $u$ in $C$.
   - By WPT, $v$ is a desc. of $u$ in the DFS $\Rightarrow$ backedge by def.
Problem Complexity

Let P be a pb. e.g. P: "sorting n #"

(a) Alg complexity: given specific alg A, to solve P (e.g. heapsort...)
what is the cost of solving P using A.

(b) pp complexity: what is the cost of solving P.

intuitively: by the best possible alg.

Decision Tree Model:

This can be used to model comparison-based alg's that work by doing
comparisons only. \( \Rightarrow \) height of tree is \( \log n \).

For each permutation \( \pi \) of \( n \); any decision tree for sorting \( n \) has
at least 1 leaf corresponding to the inverse permutation \( \pi^{-1} \) that
solves input \( \pi \). \( \Rightarrow \) the tree has at least \( n! \) leaves.

Theorem: Every comparison-based sorting alg A for sorting \( n \) requires at least
\( \Omega(n \log n) \) comparisons in the WC.

Proof:

Let \( A \) be any comp-based alg for sorting \( n \); \( \bar{A} \) be the
decision tree for \( A \).

\( \text{let } h = \text{height}(\bar{A}) \)
\( \text{note in WC, } \bar{A} \text{ takes } h \text{ comps!} \)

claim: \( h \geq \log n \)

proof of claim:

- \( \bar{A} \) has at least \( n! \) leaves (one for each permutation of \( n \))
- \( \bar{A} \) is a binary tree of height \( h \), so it has almost \( 2^h \) leaves
\( \Rightarrow 2^h \geq n! \) \( \Rightarrow h \geq \log(n!) \) which is \( \Omega(n \log n) \).

Adversary Approach

\( \text{e.g. } P = \text{"Find lonely man & wire of sets S of } n \text{ distinct elements"} \)

Naive alg: scan S twice
\[ n - 1 \text{ comps } \]
\[ n - 2 \text{ comps } \]
\[ \text{total: } 2n - 3 \text{ comps.} \]
Butler alg.

Divide S into \( \frac{n}{2} \) pairs

- to find max, min of each pair: \( \frac{n}{2} \)
- scan \( \frac{n}{2} \) times to find max: \( \frac{n}{2} - 1 \) total: \( \frac{3n}{2} - 2 \) comps.
- or scan \( \frac{n}{2} \) times to find min: \( \frac{n}{2} - 1 \)

Theorem: Every comp-based alg for min-max \( S \) takes at least \( \frac{3n}{2} - 2 \) comps in WC.

Proof of p.c.

Starting from initial state \( [n, 0, 0, 0, 0] \)

1. The alg must create \( n - 2 \) "M"s.
   To create each "M", it needs to do 1 comp in +
   \( \Rightarrow \) alg needs to do at least \( n - 2 \) comps of type #
   to create \( n - 2 \) "M"s

2. The alg must also create
   \( n - 2 \rightarrow W \) or \( L \) that becomes M
   \( W \rightarrow \) W that remains W until the end
   \( L \rightarrow \) L

\( \Rightarrow \) alg must create at least \( n - 2 \) "L" or "W"
\( \Rightarrow \) alg needs at least \( \frac{3n}{2} - 2 \) comps to create them.
Minimum Spanning Tree

A tree is a connected undirected graph with no cycles.

- A tree is undirected acyclic connected graph.
- Spanning tree of $G$ (undirected, connected) = tree $T = (V, E')$, s.t. $E \subseteq E'$
- Spanning forest is pieces of spanning tree: $F = \bigcup T_i$ s.t. $V_i \subseteq V$, $E_i \subseteq E$, $V_i \cap V_j = \emptyset$

A Minimum Spanning Tree of $G$ is s.t. $\sum w(e)$ is minimized.

- Number of spanning trees in a clique size $m$ is $m^{m-2}$.

Kruskal's MST Alg

- Sort edges alpha by weight and build forest of trees

Given a connected undirected weighted graph $G = (V, E)$

Code:

1. Build MinHeap $(E)$ # linear time
2. $Forest \leftarrow \{s_1, ..., s_m\}$ # MakeSet $(V)$
3. $MST\text{-edges} \leftarrow \emptyset$
4. While $(|MST\text{-edges}|) < m-1$, do
   1. $(u, v) \leftarrow \text{Extract-Min} (E)$
   2. $Tu \leftarrow \text{Find}(u)$; $Tv \leftarrow \text{Find}(v)$
   3. 
      - If $Tu \neq Tv$, then
        - $\text{Union}(Tu, Tv)$
        - $MST\text{-edges} \leftarrow MST\text{-edges} U \{u, v\}$
      - End

   End

Loop invariant: $MST\text{-edges}$ are contained in some MST of $G$. 

End
Traveling Salesman Problem

- **G** is completely connected w/ non-negative edge weights.
- Tour of **G**: visit every node exactly once.
- TSP tour of **G**: min-cost tour.
- TSP: to find a TSP tour of **G**.
- No brute force: exponential time $\to O(m!)$, bounded by $O(2^m)$.
- Polyhedral alg: $O(mk)$.
- TSP is NP-complete.

**DTSP**: assume that weights satisfy the triangle inequality.

- $\forall u, v, w: c(u, v) \leq c(u, w) + c(w, v)$
- DTSP is NP-complete.

Core lemma:

Let TSP be any opt tour of **G**.
Let MST be any MST of **G**.

Then $\text{cost}(\text{MST}) \leq \text{cost}(\text{TSP})$.

**Proof**:

Let $v, w, z$ be any edge from some cycle $C$.

$\forall v, w, z: c(v, w) + c(w, z) \leq c(v, z)$

By triangle inequality, cycle $C$.

Approach alg for DTSP with

1. Find MST of **G**
2. Do a full walk on it
3. Transform it into a tour $C^*$

$\text{cost}(\text{MST}) \leq \text{cost}(\text{TSP})$ gives a cycle that processes every edge exactly twice.
Time Complexity

Worst case asymptotic analysis

- $O(f(n))$:
- $\exists c, m_0 \geq 0 : f(n) \leq c \cdot g(n)$
- $\exists c, m_0 \geq 0 : f(n) \leq c \cdot g(n)$
- $\forall c, m_0, m > m_0 : f(n) \leq c \cdot g(n)$

- $f \in O(m) \implies f \not\in o(f(n)) \implies f \in O(f(n)) \implies \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Tine complexity class: $TIME(t(m)) = \{ L | L \text{ is decidable by some } O(t(m)) \text{ TM} \}$

- $\forall$ single tape TM is polynomially equivalent to multitape TM
- $\forall$ multitape TM, $T \in O(t(n)) \implies \exists$ single tape TM', $T \in O(t^2(n))$ equivalent

Non-deterministic TM has runtime $t(n) = \max$ length of branch of computation

- constrained to being a decider (arbitrarily?)
- $\forall$ TM, $T \in O(t(n)) \implies \exists$ TM', $T \in O(2O(t(n)))$ equivalent to TM

- every (strictly) polynomially bounded function $f(n) \in O(n^{log_{m}n}) \implies TIME(t(m)) \subset REG$

- $P = \bigcup_{k} TIME(n^k)$
- invariant for models of comp "measurable"
- most problems in reality aren't very poly big

- $NP = \bigcup_{k} NTIME(n^k)$
- $NP = \{ L | L \text{ has a polytime verifier} \}$
- $NP \subseteq P$?

- $PATH \in P$, $COPRIME = RELPRIME \in P$